GitHub Submission Guide

Background Information

In this course, you are required to learn how to use the distributed version control software **git** and the git repository hosting service **GitHub**. You will submit your programming assignments and I will conduct code reviews via GitHub. First, create an account, if necessary. Once you have signed up and provided me with your username via the self-diagnostic, you will receive an invitation to join our class GitHub organization, CSCI 235 – Ayzman. The following guide will get you up and running with assignment submission via GitHub.

But first, what is git?

Before I answer this question, it is important to first understand what **distributed version control** is and why it is important.

Imagine that your boss gives you an assignment. You think about. You code it up. You're done. You're happy. Your boss is happy.

Then, your boss comes up with a great new feature and tasks you with completing it. You think about it. You look at your code. You realize that making the changes will break a lot of your code. So you tentatively name the original file **awesomecode_versionA.cpp**. Then you start working on the new features in **awesomecode_versionB.cpp**. You code it up. You're done. You delete version A and keep version B. You're pleased. Your boss is pleased.

A few days later, your boss tells you that he has big plans to change the world with yet another great feature. He hires another programmer. Now, you and your new coworker need to work on this new feature together. Then, the problems begin.

You start working on the same files and you inevitably run into code conflicts. His code additions overwrite your code additions and vice versa. So you decide to name your version of the file **awesomecode_versionB_myname.cpp** and he decides to name his version **awesomecade_vursionB_hisname.cpp**, typos and all. When it comes time to merge the code together, you both pore over each other's code. Maybe you don't understand what he wrote. Maybe you do. After way too much time spent, you both figure it out and it gets done. You're not really happy. Your coworker isn't really happy. But your boss is happy. He just came up yet another great new feature. He wants to remove the first feature that you wrote yourself and replace it with something else. You stop. You think about it. You decide that you've had enough. So you quit and move to Puerto Rico.

Git is a solution to a common practice—many people changing many (of the same) files at the same time over long periods of time. Git gives you a way to easily maintain **version control** over your repository of code—working on new features, undoing older changes, keeping track of the current working version. To facilitate collaboration, git is also **distributed**, meaning that there isn't necessarily one master repository to push changes to or pull changes from. It allows developers to have their own

copies of the repository to make changes on, and it allows any developer's copy to share changes with any other developer's copy (and of course, the original). There many different kinds of version control software with their own terminologies and practices. We will be learning git because of its popularity, availability of resources, and relatively low learning curve.

What is GitHub?

GitHub maintains your git repositories in a remote, web-based location. It provides all the functionality of git with some more features of its own. There are many other web-based git repository hosting services, but GitHub is one of the more popular, well-known ones. It also has a lot of open source code!

Terminology

You can **fork** someone else's repository of code on GitHub, thereby making your own copy of it. You can also **clone** a repository from its remote, web location on GitHub down to your local machine. Your **remote** repository is considered to be the one on GitHub and your **local** repository is the one on your personal machine. You don't really edit code on GitHub. Instead, you make changes in the repository on your local machine. Once you have a finished a logical unit of work—such as fulfilling a requirement on an assignment—you **commit** those changes, or save them to your repository's git history. Each logical set of changes (decided by you) is called a **commit**. You can **push** those commits to your remote repository. This syncs your remote repository with your local one. If your remote repository changes by way of someone else (such as when a teammate pushes code to the same remote repository as you) and you want to sync your local repository with the remote one, you **pull** those changes down.

The main **branch** of your repository is usually called the **master** branch. The code on the master branch is generally considered to be unbroken and fully functional. When you work on big new changes or features, you often create a new **branch** off of the **master** branch, and make your commits on this new branch. Then, once you are satisfied that the feature is complete, you can **merge** those commits into the master branch. When you are working with a team, you often want people to review your changes before they become included in the master branch. Therefore, you can make a **pull request** on GitHub, requesting that the changes you made on your experimental feature branch be merged into the master branch. While the pull request is open, anyone can comment on the code and ask you to make changes. They might see potential bugs or stylistic errors or simply ask for clarification on what the code is doing. Once the reviewer is satisfied, he/she **accepts** the pull request, thereby completing the merge into the master branch. Do not confuse **making pull requests** with **pulling changes** from the remote repository to your local repository. They are completely different actions.

Course Specific Details

When you first visit this course's GitHub organization page, you'll see several repositories. You'll see some public repositories, visible to anyone viewing the organization page. This includes any useful repository that the world might benefit from seeing or knowing about, such as **Tech_Resources**. You'll also see some private repositories, such as assignment repositories with canonical instructions and starter resources. These are visible to anyone in the class who wishes to reference them. You will also have your own personal repository for each assignment available, visible only to you. The naming

convention for these personal repositories is ***githubusername-canonicalrepositoryname***, where ***githubusername*** is your own GitHub username and ***canonicalrepositoryname*** is the name of the repository with the canonical assignment instructions. These personal repositories are where you will be making changes, or effectively "submitting your assignments." To start, each personal assignment repository should have the exact same content as its canonical assignment repository counterpart.

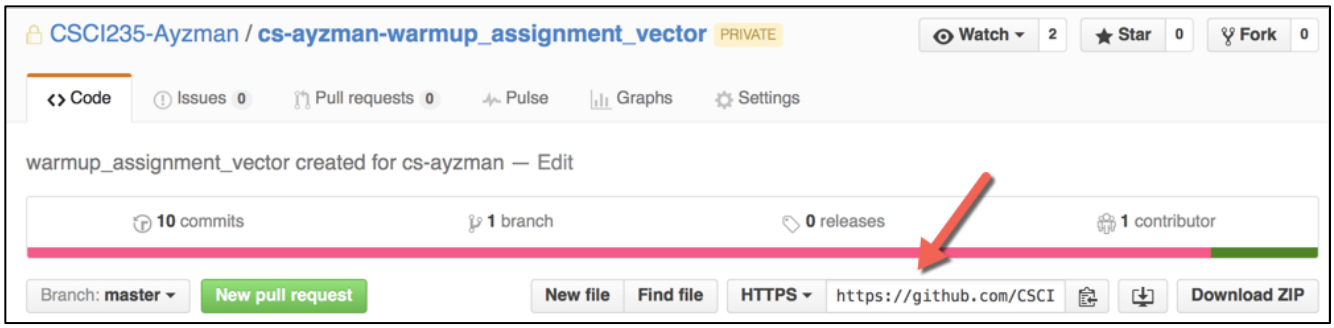
Submission Walkthrough

Begin by opening the command prompt, terminal, or powershell on your local machine. Create a new directory/folder where your assignment code will live. If necessary, make sure that you install git and that you go through the steps of properly setting it up.

Start by cloning your personal assignment repository down to your local machine. Run the command:

> **git clone *repository_url***

where ***repository_url*** is the url supplied on the GitHub repository's main page.



Before you start coding, confirm that you're going to be doing work on your master branch.

> **git checkout master**

Create a branch called **assignment**.

> **git branch assignment**
> **git checkout assignment**

or, simply do:

> **git checkout -b assignment**

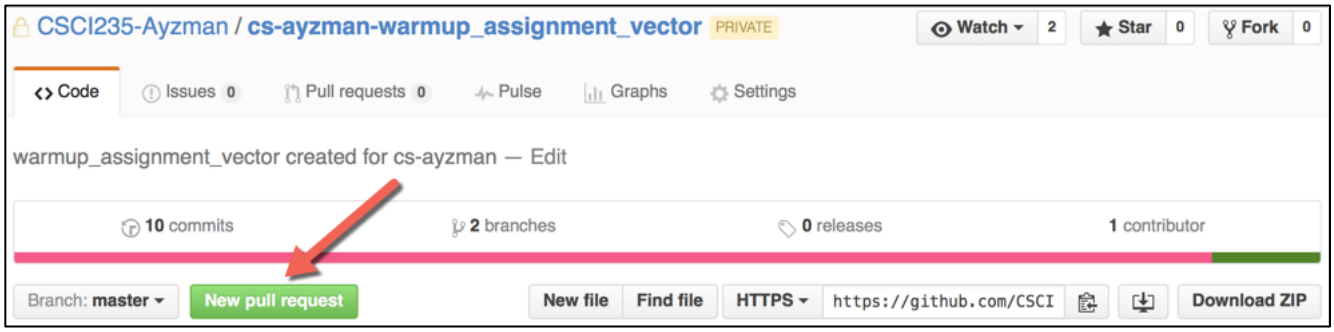
As you work on your assignment, make commits every time you feel that you have completed a logical unit of work.

> **git add <files relevant to this commit>**
> **git commit -m "<commit message>"**

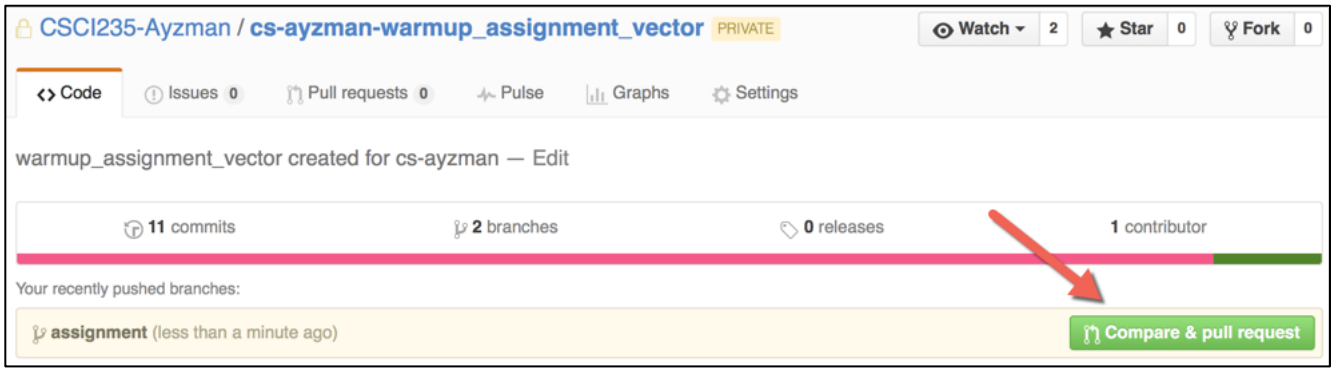
where **<files relevant to this commit>** is a whitespace separated list of files that you want to commit and **<commit message>** is a string that explains the changes made in this commit. Ever so often, push your local commits up to your remote repository.

> **git push origin assignment**

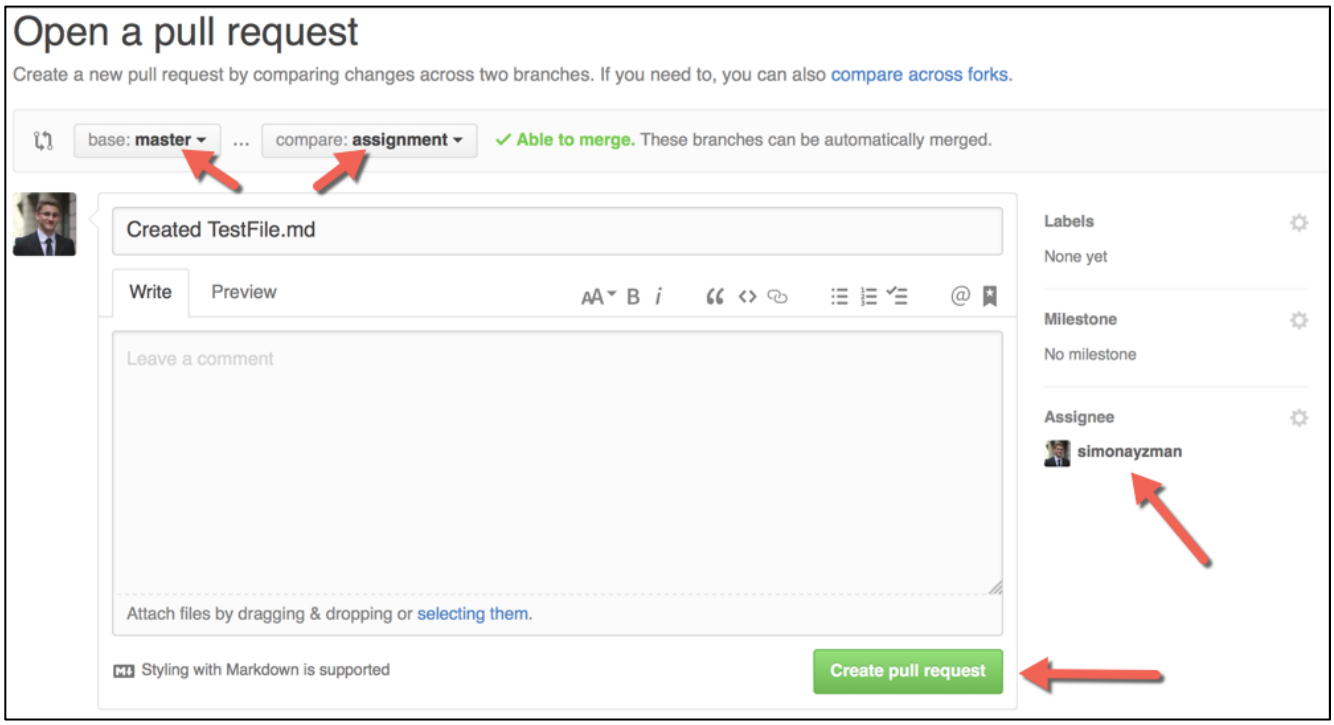
Once your assignment is finished and fully pushed to your remote repository, go online to your GitHub repository and create a new pull request.



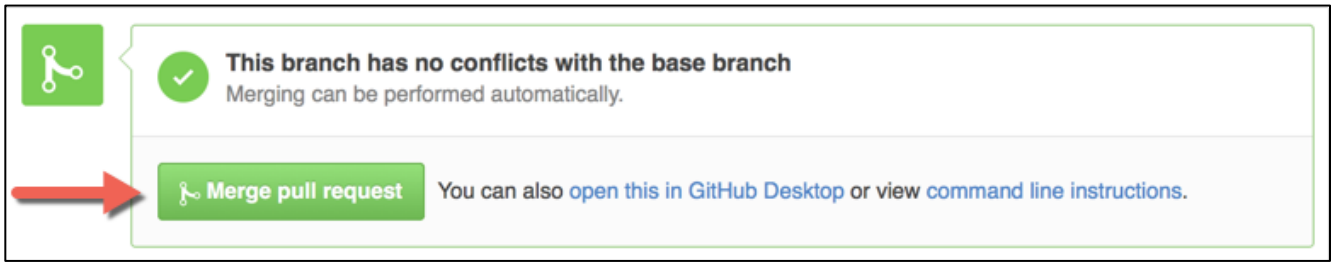
Alternatively, you might see this:



The "base" branch (where the new changes will be added) should be **master**, and your "compare" branch (where the new changes are coming from) should be **assignment**. Be sure to assign **simonayzman** as a reviewer.

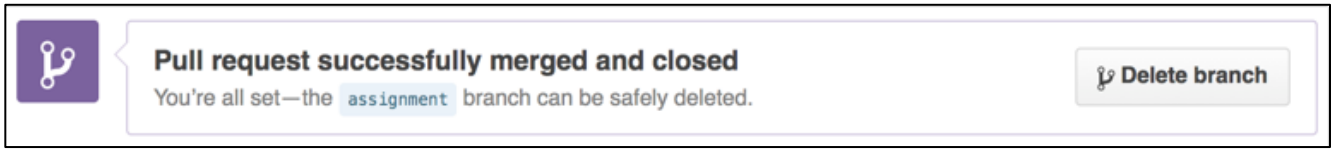


Once your pull request is created, you will see a button to merge in the pull request. Do not merge the pull request yourself!



In a real world scenario, a software engineer would review the code in your pull request and make comments. The reviewer might not be satisfied with some aspect your code and may request that you change something before he/she accepts your pull request. This would give you the opportunity to make the requested changes on your branch via new commits. When you push your changes to the remote repository, the pull request will automatically update with the new commits. This comment/update back-and-forth may continue for several cycles. Once the reviewer is satisfied with your changes, only then will he/she accept your pull request. Alternatively, the reviewer might close the request altogether. Ultimately, you would almost never be allowed to accept your own pull requests.

Congratulations! You've submitted your assignment. In the process, you learned how to use a powerful distributed version control toolset. You're well on your way to becoming a git and GitHub master (no pun intended). If you submit the assignment before the deadline and I provide you with comments in time before the deadline, you will have an opportunity to update your pull request! Once the deadline hits, I will merge in your pull request and you will no longer be able to push new changes to it. The current state of your repository's master branch will be the version I will compile, run, and grade.



I will review your code in much the same way a software engineer in a real world context would. Note that I can comment on your pull request's code on GitHub regardless of the pull request's status—whether it is closed altogether, open and awaiting review, or accepted and merged into the base branch. I hope the comments provide you with good feedback about how to improve your code quality!

Learning Resources

For a more structured dive into git and GitHub, check out the following resources (themselves in a GitHub repository! How meta!). Going through the motions is one thing, but learning how to effectively use git will encourage you to think more logically, bolster your developer toolset, and considerably increase your overall value as a software engineer.